LE CONTROLE D'ERREURS
LES CODES AUTOVERIFICATEURS OU AUTOCORRECTEURS

Les codes de blocs

Le principe employé dans les codes de blocs consiste à construire le mot de code en « sectionnant » l'information utile en blocs de longueur fixe et en ajoutant à chaque bloc, ainsi obtenu, des bits de contrôle supplémentaires (bits de redondance). On crée alors un code de blocs, où seules certaines des combinaisons possibles sont valides et forment l'ensemble des mots du code. À la réception deux cas peuvent se présenter :

- Le mot de n bits reçu est un mot de départ peut être reconstitué,
- Le mot de n bits reçu ne correspond pas à un mot de code et le récepteur peut alors soit retrouver le bloc original (codes autocorrecteurs) soit s'il ne le peut pas redemander la transmission du message précédent (codes vérificateurs).

L'efficacité d'un tel code sera d'autant meilleure que les mots qui le consistent seront distincts les uns et les autres. On définit ainsi une distance entre les différents mots qui composent le code, **distance de Hamming**, correspondant au **nombre de bits qui varient entre deux mots successifs du code.**

Plus la distance est importante et plus efficace sera le code.

Parmi les codes de blocs, on rencontre communément :

- le contrôle de parité verticale, parfois aussi nommé VRC (Vertical Redundancy Checking), dont le principe de la parité a déjà été vu (nb pair de 1),
- le contrôle de parité longitudinale, ou LRC (Longitudinal Redundancy Checking)
- ainsi que divers codes, dit i parmi n, généralement associés à une information de redondance.

En utilisant ces codes de blocs, et le contrôle de parité, il est possible d'assurer une vérification dite par **parités croisées** ou **LRC/VRC** qui, en augmentant la distance de Hamming, assure une meilleure détection et la correction de certaines erreurs. Il convient pour cela de regrouper les caractères de blocs et d'ajouter à la fin de chaque bloc un caractère supplémentaire dit LRC qui se combine au contrôle VRC.

On souhaite transmettre les caractères P A G en code ASCII.

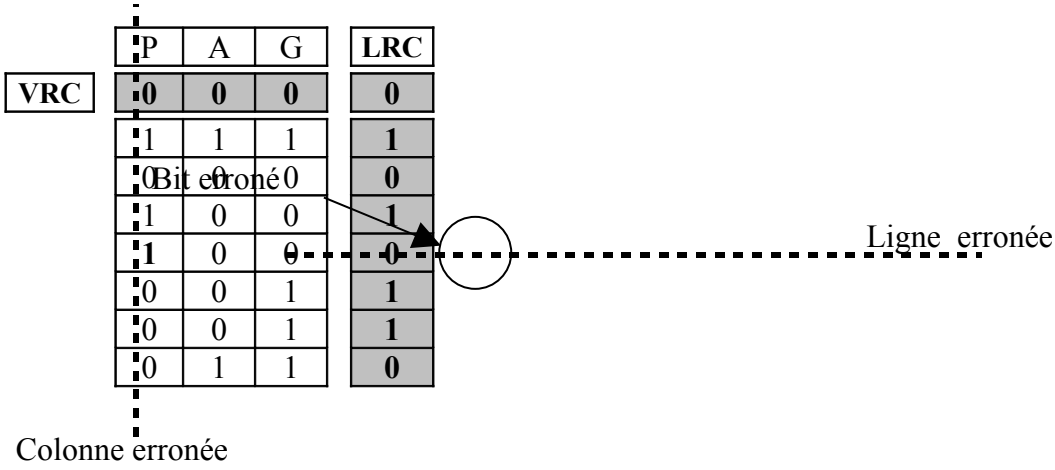
	P	A	G	**LRC**
VRC	**0**	**0**	**0**	**0**
	1	1	1	**1**
	0	0	0	**0**
	1	0	0	**1**
	0	0	0	**0**
	0	0	1	**1**
	0	0	1	**1**
	0	1	1	**0**
	50_{16}	41_{16}	47_{16}	

← Parité croisée

En fait le caractère VRC est un contrôle de parité verticale tandis que le LRC est un contrôle de parité horizontale (longitudinale).

Un tel code détecte donc toutes les erreurs simples, doubles ou triples et peut corriger toutes les erreurs simples. Ainsi, en reprenant les caractères précédemment transmis, et si on considère qu'une erreur de transmission a affecté un des bits, voyons comment va procéder le système pour **détecter et corriger une erreur simple** telle que ci-dessous.

La résolution de ce problème est en fait relativement simple. Il suffit en effet de s'assurer dans un premier temps de ce que **la parité croisée** vérifie bien les codes LRC et VRC. Ici, la parité croisée nous indique un bit à 0. La parité semble donc bien respectée sur les bits de contrôle VRC et sur les bits de contrôle LRC. L'erreur ne vient donc pas d'eux à priori.
En revanche, si l'on vérifie les parités LRC, on peut aisément détecter la ligne où s'est produite l'erreur. En vérifiant les parités VRC on détecte facilement la colonne erronée. L'intersection de cette ligne et de cette colonne nous permet alors de retrouver le bit erroné et partant, de le corriger.



Les codes cycliques

Les codes cycliques, aussi appelés CRC (Cyclic Redundancy Codes) ou codes polynomiaux, sont des codes de blocs d'un type particulier, très utilisés du fait de leur facilité de mise en œuvre matérielle. Ils sont basés sur l'utilisation d'un polynôme générateur G(x) qui considère que toute information de n bits peut être transcrite sous une forme polynomiale.
En effet, à l'information binaire 10111, on peut associer le polynôme $X^4 + X^2 + X^1 + X^0$.
C'est à dire :
1 0 1 1 1
$X^4 + X^2 + X^1 + X^0$.

À partir d'une information de départ I(x) de i bits on va alors construire une information redondante R(x) de r bits et l'émettre à la suite de I(x), de telle sorte que le polynôme résultant N(x) soit divisible par le polynôme générateur G(x).
À la réception, on divise le polynôme N(x) reçu par le même polynôme G(x) et le reste de cette division doit alors être nul s'il y a pas eu d'erreur de transmission.

Soit le message 1 0 0 1 1 0 1 1 (i=8) que l'on peut traduire par le polynôme I(x) :
$1 \times X^7 + 0 \times X^6 + 0 \times X^5 + 1 \times X^4 + 1 \times X^3 + 0 \times X^2 + 1 \times X^1 + 1 \times X^0$
ou plus simplement : $I(x) = X^7 + X^4 + X^3 + X^1 + X^0$

Le polynôme générateur choisi, ici, est arbitrairement $G(x) = X^3 + 1$ avec r=4
(r étant égal au nombre de bits qui constituent le polynôme générateur).
On multiplie alors I(x) par le polynôme G(x) – 1.

1 0 0 1 1 0 1 1 **I(x)**
x 1 0 0 0 **G(x) – 1 = $(X^3 + 1) = X^3$**
---
1 0 0 1 1 0 1 1 0 0 0

On effectue ensuite la division du polynôme ainsi obtenu par le polynôme générateur
soit $X^{10} + X^7 + X^6 + X^4 + X^3$ à diviser par $X^3 + 1$.

Soit en binaire :

```
 1 0 0 1 1 0 1 1 0 0 0  |  1 0 0 1
 1 0 0 1                |------------
 ---------              |  1 0 0 0 1 0 0 1
       0 1 0 1 1        |
         1 0 0 1        |
         --------       |
             1 0 0 0 0  |
               1 0 0 1  |
             ---------- |
      reste    0 1 1 1  |
```

Il suffit donc d'ajouter le reste trouvé au dividende I(x) initial (10011011) pour obtenir un dividende divisible par le diviseur (ce qui est le but recherché) :

```
      1 0 0 1 1 0 1 1      I(x)
            0 1 1 1        r
-------------------------
      1 0 1 0 0 0 1 0      I'(x)
```

L'information transmise sera alors I'(x) soit 10100010, à laquelle il faut adjoindre le reste r, soit en définitive : 10100010 – 0111

À la réception, le système divise le I'(x) reçu par le polynôme générateur (et le reste de cette division doit alors être nul si la transmission s'est bien passée).

```
 1 0 1 0 0 0 1 0   I'(x)  |  1 0 0 1        G(x)
 1 0 0 1                  |-------------
 ---------                |  1 0 0 1 0
       1 0 0 1            |
       1 0 0 1            |
       --------           |
             0 0          |  → le reste est égal à zéro, donc la transmission s'est correctement effectuée.
```

Il ne lui reste plus maintenant qu'à retrancher le r reçu de I'(x) pour retrouver l'information émise.

```
      1 0 1 0 0 0 1 0      I'x)
            0 1 1 1        r
-------------------------
      1 0 0 1 1 0 1 1      I(x)    → Information émise
```

Le choix d'un bon polynôme générateur est important si l'on veut détecter un maximum d'erreurs. Un polynôme très utilisé, normalisé par le CCITT est : $X^{16} + X^{12} + X^5 + 1$.
Ce polynôme permet de détecter :

- 100 % des erreurs simples ou doubles,
- 100 % des erreurs sur un nombre impair de bits,
- 100 % des paquets d'erreurs d'une longueur ≥ 16 bits,
- 99,99 % des paquets d'erreurs d'une longueur > 18 bits.

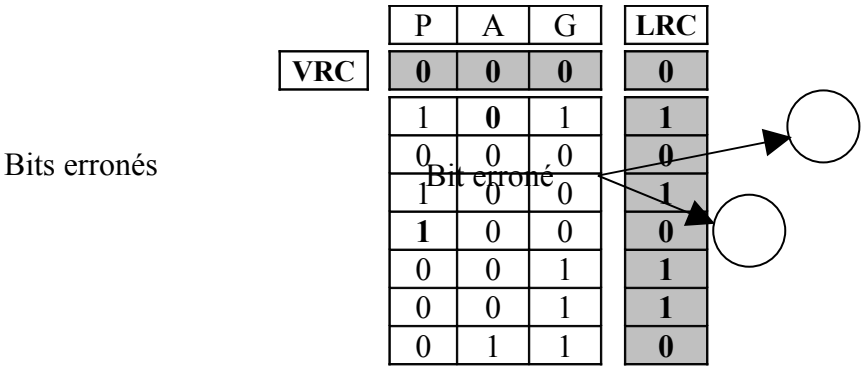
Un tel code permet donc de diviser le taux d'erreur moyen par 100 ou 1000, suivant le type d'erreur, et n'introduit qu'une redondance de 16 bits pour des messages dont la taille courante est de l'ordre de 1000 bits. Dans la pratique, ces codes seront utilisés en télétransmission entre deux appareils reliés par un réseau téléinformatique.

EXERCICE 1

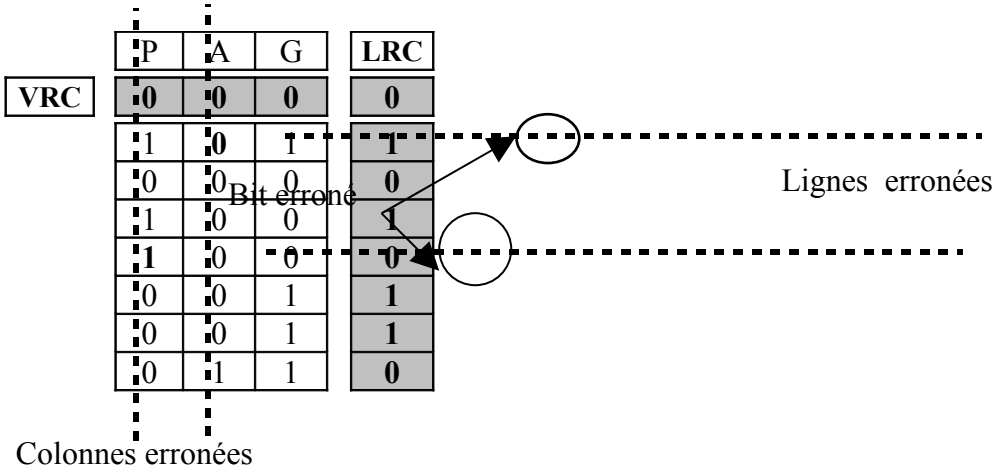
Réfléchir à la manière dont le système peut tenter de s'en sortir à l'aide des parités longitudinales et verticales, lorsque deux bits sont erronés à la réception des données.

Pourquoi ne peut-on plus corriger l'erreur mais seulement en détecter la présence ?

Prenons un exemple de tableau où se produisent deux erreurs. La parité croisée vérifie, dans notre exemple, les contrôles VRC et LRC ce qui permet de penser qu'à priori l'erreur ne vient pas des contrôles verticaux ou longitudinaux eux-mêmes mais des bits de données proprement dits.



Quand on vérifie les parités LRC et VRC on détecte la présence de 2 lignes et de 2 colonnes où se sont produites des erreurs ce qui se traduit en fait par 4 points d'intersections possibles, ainsi que le montre le schéma ci-après.



Ceci met le système dans l'impossibilité de corriger 2 seulement de ces bits. En effet lesquels choisir ? Ceux correspondant à l'erreur réelle ou bien ceux correspondant à l'erreur possible ?

Devant cette indétermination, on en est réduit à une seule possibilité. On détecte bien qu'il y a eu une erreur, mais on est dans l'impossibilité de corriger cette erreur. Cette technique est donc simple mais pas forcément la plus fiable.

EXERCICE 2

En utilisant le code générateur $X^3 + 1$ retrouver les valeurs hexadécimales qui seront réellement transmises pour faire passer la suite binaire 10110110.

La première chose à faire consiste à multiplier la série binaire par le polynôme générateur G(x) – 1. Rappelons que G(x) correspond ici, et arbitrairement, à la série binaire 1001. Le résultat de cette première opération sera donc : 1011 0110 x 1000 = 101 1011 0000

Il nous faut ensuite procéder à la division de cette suite binaire par le polynôme générateur

10110110000 / 1001 = 10100001 le reste de la division est : 111

Il faut alors ajouter ce reste à la suite binaire initiale soit : 10110110 + 0111 = 10111101

La suite binaire transmise sera donc finalement 10111101, à laquelle nous « accolerons » le reste trouvé soit : 10111010 111

Ce qui, une fois transcrit en hexadécimal donne $BD7_{16}$

Détails exercice 2

```
      1 0 1 1 0 1 1 0        I(x)
          x 1 0 0 0          G(x) – 1 = (X³ + 1) = X³
--------------------------
 1 0 1 1 0 1 1 0 0 0 0
```

On effectue ensuite la division du polynôme ainsi obtenu par le polynôme générateur
soit $X^{10} + X^8 + X^7 + X^5 + X^4$ à diviser par $X^3 + 1$.
Soit en binaire :

```
 1 0 1 1 0 1 1 0 0 0 0  |  1 0 0 1
 1 0 0 1                |  ---------
 ---------              |  1 0 1 0 0 0 0 1
     1 0 0 1            |
     1 0 0 1            |
     --------           |
           0 1 0 0 0 0  |
               1 0 0 1  |
             ---------- |
    reste      0 1 1 1  |
```

Il suffit donc d'ajouter le reste trouvé au dividende I(x) initial (10110110) pour obtenir un dividende divisible par le diviseur (ce qui est le but recherché) :

```
      1 0 1 1 0 1 1 0        I(x)
            0 1 1 1          r
--------------------------
      1 0 1 1 1 1 0 1        I'(x)
```

L'information transmise sera alors I'(x) soit 10111101, à laquelle il faut adjoindre le reste r, soit en définitive : 10111101 – 0111

À la réception, le système divise le I'(x) reçu par le polynôme générateur (et le reste de cette division doit alors être nul si la transmission s'est bien passée).

```
 1 0 1 1 1 1 0 1   I'(x)  |  1 0 0 1      G(x)
 1 0 0 1                  |  ---------
 ---------                |  1 0 1 0 1
     1 0 1 1              |
     1 0 0 1              |
      --------            |
         1 0 0 1          |
         1 0 0 1          |
         --------         |
               0
```

→ le reste est égal à zéro, donc la transmission s'est correctement effectuée.

Il ne lui reste plus maintenant qu'à retrancher le r reçu de I'(x) pour retrouver l'information émise.

```
      1 0 1 1 1 1 0 1        I'x)
            0 1 1 1          r
--------------------------
      1 0 1 1 0 1 1 0        I(x)
```

→ Information émise